

```

-- Selecter
-- version4 - 3/5/2004
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
library synplify;
use synplify.attributes.all;

entity Selecter is port
  (
    clock          : in std_logic;
    resetn         : in std_logic;
    brd_ad         : in std_logic_vector(7 downto 0);
    HUB_responding : out std_logic;

    -- SCC signals
    parity_error   : in std_logic;
    overflow        : in std_logic;
    TXrdy          : in std_logic; -- serial data can be sent to UART, TXrdy
    receive_full    : in std_logic; -- serial data_in available from UART, recieve_full
    SCC_Data_RCVD  : in std_logic_vector(7 downto 0); -- SCC_SCC_Data_RCVD[7..0]
    WEn            : out std_logic; -- load serial data_out into UART, WEn
    OEn            : out std_logic; -- registered serial data_in from UART, OEn
    SCC_Data_TRXD  : out std_logic_vector(7 downto 0); -- SCC_data_in[7..0]

    state_out       : out std_logic_vector(3 downto 0);
    CCM_Selector   : out std_logic_vector(8 downto 0);
    CCM_reset       : out std_logic;
    break           : out std_logic);

end Selecter;

architecture behave_rtl of Selecter is
attribute syn_radhardlevel of behave_rtl : architecture is "tmr";

type state_values is (st0, st1, st2, st3, st4, st5, st6, st7, st8, st9, st10, st11, st12);
signal pres_state      : state_values;

signal STATUS_REGISTER_63 : std_logic_vector(7 downto 0);

signal load_timer       : std_logic;
signal timer_en         : std_logic;
signal timed_out        : std_logic;
signal timer             : std_logic_vector(14 downto 0);
signal timer_value       : std_logic_vector(14 downto 0);

signal CCM_Selector_temp : std_logic_vector(8 downto 0);
signal CCM_reset_enable  : std_logic;

begin
  -- fsm register
  state_reg: process (clock, resetn)
  begin
    if (resetn = '0') then
      pres_state      <= st0;
      break           <= '1';
      SCC_Data_TRXD  <= "00000000";
      WEn            <= '1';
      OEn            <= '1';
      CCM_reset_enable <= '0';
      CCM_reset       <= '1';
      CCM_Selector   <= "00000000";
    end if;
  end process state_reg;
end;

```

```

CCM_Selector_temp      <= "000000000";
load_timer             <= '0';
timed_out              <= '0';
timer_en               <= '0';
timer                 <= "001001011000000";
HUB_responding         <= '0';
STATUS_REGISTER_63     <= "00000000";
timer_value <= "001001011000000";

elsif clock'event AND clock = '1' then

----- 1001011000000
if load_timer = '1' then
    timer <= timer_value;-- "1001011000000"=120us
elsif timer_en = '1' then
    timer <= timer - 1;
else
    timer <= timer;
end if;

if timer = "000000000000000" then
    timed_out <= '1';
else
    timed_out <= '0';
end if;
-----

case pres_state is
when st0 =>
    state_out <= ("0000");
    WEn <= '1';
    OEn <= '1';
    CCM_reset      <= '1';
    CCM_reset_enable <= '0';
    break          <= '1';
    timer_en       <= '0';
    HUB_responding <= '0';
    CCM_Selector_temp <= "000000000";
    if receive_full = '1' then
        OEn <= '0'; -- register data from SCC
        pres_state <= st1; -- write
    else -- recieve_full = 0 so stay at st0
        pres_state <= st0;
    end if;

when st1 =>
    state_out <= ("0001");
    OEn <= '1'; -- reset the receiver buffer
    load_timer <= '1';
    Case SCC_Data_RCVD(7 downto 6) is -- select port to enable
        when "00" => timer_value <= "001001011000000";

            if SCC_Data_RCVD(5 downto 0) = "111111" then
                STATUS_REGISTER_63(7) <= parity_error;
                STATUS_REGISTER_63(6) <= overflow;

                    pres_state <= st2; -- write Custom HUB with next byte
                else
                    pres_state <= st9; -- error in writing the hub
                end if;
            when "01" => timer_value <= "000111000010000"; pres_state <= st9;
            when "10" => timer_value <= "001101010010000"; pres_state <= st9;
            when "11" => timer_value <= "111110100000000"; pres_state <= st9;
            when others => pres_state <= st0;

--90us
--170us
--800us

```

```

    end case;

when st2 => -- Wait for next byte to select port and or reset that port
    state_out <= ("0010");
    if (receive_full = '1') then
        OEn <= '0';
        timer_en <= '0';
        pres_state <= st3;
    else -- wait an amount of time
        timer_en <= '1';
        if timed_out = '1' then -- second byte not present
            pres_state <= st0; -- so start over
        else
            pres_state <= st2;
        end if;
    end if;

-- Write single byte sequence
when st3 => -- enable port and/or reset
    state_out <= ("0011");
    STATUS_REGISTER_63(5 downto 0) <= SCC_Data_RCVD(5 downto 0);
    STATUS_REGISTER_63(7) <= parity_error or STATUS_REGISTER_63(7);
    STATUS_REGISTER_63(6) <= overflow or STATUS_REGISTER_63(6);

    if      SCC_Data_RCVD(5) = '1' then -- if bit 5 on then enable reset to
port
        CCM_reset_enable <= '1';
    else
        CCM_reset_enable <= '0';
    end if;
    if      SCC_Data_RCVD(6) = '1' then -- if bit 6 on then send break to po
rt
        break <= '0';
    else
        break <= '1';
    end if;

Case SCC_Data_RCVD(3 downto 0) is -- select port to enable
    when "0001" => CCM_Selector_temp <= "000000001";
    when "0010" => CCM_Selector_temp <= "000000010";
    when "0011" => CCM_Selector_temp <= "000000100";
    when "0100" => CCM_Selector_temp <= "000001000";
    when "0101" => CCM_Selector_temp <= "000010000";
    when "0110" => CCM_Selector_temp <= "000100000";
    when "0111" => CCM_Selector_temp <= "001000000";
    when "1000" => CCM_Selector_temp <= "010000000";
    when "1001" => CCM_Selector_temp <= "100000000";
    when "1111" => CCM_Selector_temp <= "111111111";
    when others => CCM_Selector_temp <= "000000000";
end case;
OEn <= '1';           -- reset the receiver buffer
pres_state <= st4;

when st4 => -- return board address
    state_out <= ("0100");
    if (TXrdy = '1') then
        SCC_Data_TRXD <= brd_ad;
        HUB_responding <= '1';
        WEn <= '0'; -- register Board Address in the UART to transmit
        pres_state <= st5;
    else
        pres_state <= st4;
    end if;

when st5 => -- WEn needs to be reset
    state_out <= ("0101");

```

```

        if (TXrdy = '0') then      -- wait until it goes low
            pres_state <= st6;
            WEn <= '1';           -- set back to normal
        else
            pres_state <= st5;
        end if;

when st6 => -- return status,
    state_out <= ("0110");
    if (TXrdy = '1') then
        SCC_Data_TRXD <= STATUS_REGISTER_63;
        WEn <= '0'; -- register status data in the UART to transmit

        pres_state <= st7;
    else
        pres_state <= st6;
    end if;

when st7 => -- WEN needs to be reset
    state_out <= ("0111");
    load_timer <= '1';
    if (TXrdy = '0') then      -- wait until it goes low
        pres_state <= st8;
        WEn <= '1';           -- set back to normal
    else
        pres_state <= st7;
    end if;

when st8 => -- WEN needs to be reset
    state_out <= ("1000");
    CCM_Selector <= CCM_Selector_temp; --set output port
    load_timer <= '1';
    if (TXrdy = '1') then      -- wait until it goes low
        pres_state <= st9;
    else
        pres_state <= st8;
    end if;

-- need to wait until transfer of status is over and then switch to RBX communication
-- should be 11 bits of data

when st9 => -- wait for 11 bits to transfer or for RBX Communication to finish
    state_out <= ("1001");
    timer_en <= '1';
    load_timer <= '0';
    if timed_out = '1' then --
        pres_state <= st10; --
    else
        pres_state <= st9;
    end if;

when st10 => -- set output port
    state_out <= ("1010");
    timer_en <= '0';
    load_timer <= '1';
    timer_value <= "1111101000000000"; -- use 800us for reset pulse
    HUB_responding <= '0';
    pres_state <= st11;

when st11 => -- WEn needs to be reset, send reset to port selected
    state_out <= ("1011");
    if CCM_reset_enable = '1' then
        CCM_Reset <= '0';
        pres_state <= st12;
    else
        CCM_Reset <= '1';
        pres_state <= st0;
    end if;

```

```
when st12 =>
    state_out <= ("1100");
    timer_en <= '1';
    load_timer <= '0';
    if timed_out = '1' then -- give the reset a large pulse.
        pres_state <= st0;
    else
        pres_state <= st12;
    end if;

end case;
end if;
end process;
end behave_rtl;
```